LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-PROC-648476

FUDGE: A Toolkit for Nuclear Data Management and Processing

B. R. Beck, C. M. Mattoon

January 14, 2014

2014 ANS Annual Meeting
Reno, NV, United States
June 15, 2014 through June 19, 2014

Disclaimer

FUDGE: A Toolkit for Nuclear Data Management and Processing

B. R. Beck and C. M. Mattoon

Lawrence Livermore National Laboratory, L-059, P.O. Box 800, Livermore, CA 94551
beck6@llnl.gov and mattoon1@llnl.gov

INTRODUCTION

For over 50 years the Evaluated Nuclear Data File (ENDF) [1] format has been used for storing and sharing evaluated nuclear reaction data. Processing codes like NJOY [2] and AMPX [3] are used to transform ENDF formatted data into forms useful for reactor, medical and other transport codes. For example, NJOY can be used to produce ACE files used by the transport code MCNP [4]. With all of its advantages, the ENDF format has limitations as it was developed in the era of punchcards, small (kilo-bytes) memory computers and limited storage capacity that restricted its design. Furthermore, computer software technology has made significant advances over the last 50 years. To address ENDF's limitations, an international committee (OECD/NEA WPEC sub-group 38) [5] has been working since December 2012 to design a modern format, tentatively called GND, to replace the ENDF format.

To complement the new format, the nuclear data group at Lawrence Livermore National Laboratory (LLNL) is modifying its nuclear data management infrastructure (called FUDGE) to support reading/writing, plotting, manipulating and processing nuclear data stored in the GND format. Over the next few years, FUDGE will probably be the only infrastructure that is capable of supporting the new format. FUDGE will allow the conversion of data in the new format to the ENDF format so that data in the new format can be processed by codes that only support ENDF.

FUDGE is freely available and can be downloaded from https://ndclx4.bnl.gov/gf/project/gnd/.

BRIEF HISTORY OF ENDL, FUDGE AND GND

While the rest of the world used the ENDF format for storing nuclear reaction data, LLNL had its own format called ENDL (Evaluated Nuclear Data Library) [6]. The ENDL format was developed several years before ENDF and has many of the limitations found in ENDF and a few of its own. In addition, by sticking to the ENDL format, LLNL has in many ways isolated itself from the rest of the nuclear data community, requiring it to develop its own data and processing codes.

Around 2001 LLNL started the development of a user-friendly toolkit called FUDGE (For Updating Data and Generating ENDL) to simplify the reading/writing, plotting, modifying and processing ENDL data. The user interface to FUDGE is a suite of Python modules. Computationally intensive calculations are coded in lower level languages like C and C++; however, users do not need to interact with these codes as Python wrappers are provided. Python was chosen since it is object oriented and has a large set of external modules (e.g., Gnuplot and matplotlib for plotting data).

Around 2001 LLNL also started developing a code dubbed FETE (From ENDF To ENDL) [7] to translate ENDF data into ENDL data. While this project achieved its goal, it was realized that the ENDL format had too many limitations to handle all data coming from ENDF with full fidelity. Hence, a search for a better format was started around 2005. With limitation also discovered in the ENDF format, LLNL decided to develop a new nuclear data format called GND (Generalized Nuclear Data) [8] and to modify its FUDGE toolkit to support it.

FUDGE TOOLKIT

Most Python modules in the FUDGE toolkit (herein called FUDGE) contain classes for a particular type of data. For example, the crossSection.py module contains classes for various representations for cross section data (e.g., pointwise (ENDF MF 3), resonance with background (ENDF MF 2 and 3)). The following sections outline the major parts of FUDGE.

Reading/Writing Data

As FUDGE was originally written for the ENDL format, it still maintains the classes and their methods for ENDL formatted data including reading and writing. FUDGE now also supports reading and writing of a GND/XML[a] file.

Data in most (see end of section) of the ENDF formatted files can be converted to GND and written out as a GND/XML file with the following Python coding[b]:

```
>>> from fudge.legacy.converting import \
                    endfFileToGND as E2GND
>>> rce = E2GND.endfFileToGND( 'n-92_U_239.endf' )
>>> reactionSuite = rce['reactionSuite']
>>> reactionSuite.saveToFile( 'n-92_U_239.GND' )
```

where the ENDF file to translate is named 'n-92_U_239.endf'. The function E2GND.endfFileToGND returns a dictionary containing 3 objects. One object with key "reactionSuite" contains the data in the ENDF file,

except for the covariance data, as an instance of a FUDGE 'reactionSuite' class. A 'reactionSuite' contains a list of reactions for a projectile hitting a target as well as other data (i.e., most of the data in an ENDF material). Another object with key "covarianceSuite" contains the covariance data. And the last object with key "errors" contains a list of errors in the ENDF file discovered by the parser.

In addition to translating an ENDF formatted file to GND, FUDGE supports the translation of a GND evaluation to an ENDF formatted file. This is important since it will be several years before WPEC sub-group 38 completes the design of GND and several more years before many of the processing codes will support GND. For the above example, this would be

```
>>> fOut = open( '92_U_239.endf.2', 'w' )
>>> fOut.write( reactionSuite.toENDF6( ... ) )
>>> fOut.close( )
```

where, for brevity, the arguments to the method 'toENDF6' are not shown.

For ENDF-VII.1, FUDGE can translate the following libraries (with exception of a few charged-particle files that have data issues):

neutrons/ protons/ deuterons/
tritons/ helium3s/ gammas/
standards/

Translators for the following libraries exist but have not been integrated into FUDGE yet:

nfy/ sfy/ thermal_scatt/

While translation of the following libraries is still needed:

decay/ electrons/ photoat/
atomic_relax/

Data Representations and Plotting

FUDGE (and GND) supports all the data representation of the ENDL and ENDF formats. In ENDL the data representations are all pointwise forms (e.g., a cross section, $\sigma(E)$, is a list of (E_i, σ_i) pairs). In addition to pointwise forms, ENDF supports some functional forms for the data (e.g., N-Body Phase-Space Distributions (ENDF MF 6 LAW 6)).

All pointwise data classes in FUDGE have a 'plot' method. Using the 'n + ^{239}U' evaluation from the ENDF/B-VII.1 library [9] for the example above and reconstructing the cross section from resonance parameter data, the FUDGE coding for plotting the elastic cross section is:

```
>>> reactionSuite.reconstructResonances( )
>>> elastic = reactionSuite.getReaction( 'elastic' )
>>> crossSection = elastic. crossSection
>>> XSec = crossSection.toPointwise_withLinearXYs( )
>>> XSec.plot( )
```

The plot is shown in Figure. 1. In addition to a plot, a dialog window (not shown) allows one to change plot parameter such as axis limits and labels.

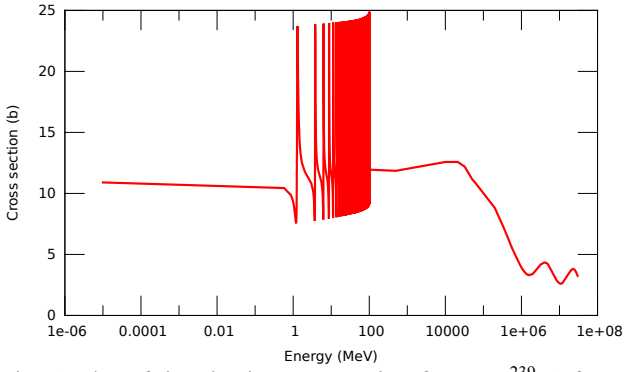

Fig. 1. Plot of the elastic cross section for 'n + ^{239}U' from ENDF/B-VII.1.

All data classes in FUDGE have a method called "toPointwise_withLinearXYs" that converts the data to a "linear" pointwise representation, allowing for plotting of functional data forms. For example, a fission neutron distribution represented with the 'WattSpectrum' class (ENDF MF 5 LF 11) can be plotted by first converting it to a pointwise representation.

Modifying Data

One of the original goals for FUDGE was to simplify the modifying and processing of ENDL data so that sensitivity studies can be easily be performed. Many of FUDGE data classes contain methods for simplifying the modification of data. For example, the 1-d (i.e., any function of the form y(x) like $\sigma(E)$) pointwise data class dubbed XYs has methods for the operators '+', '-', '*' and '/'. These operators support the second operand being a number or another XYs instance (e.g., if 'a' is an XYs instance, 'c = a + b' is supported when 'b' is a number or another XYs instance). Since ENDF data are usually not on a common energy-grid (i.e., x-grid for the XYs class), these operators first create a common x-grid from the two operands and then interpolate their y-values. This allows, as example, for the addition of two pointwise ENDF cross sections.

For example, if the uncertainty as a function of energy for the cross section for the fission reaction for 'n + ^{239}U' is known, what is the k_{eff} uncertainty for a critical assembly? One solution is to have a transport code that

propagates uncertainties. Another solution, and the one FUDGE was designed for, is to sample 'realization's of the data. That is, use Monte Carlo to sample from a data's uncertainty and add this to the data (one realization or possible curve given the data and its uncertainty). As example, if 'unc' is a Python variable containing the energy-dependent fractional uncertainty for the 'XSec' data in the prior example, with energy dependence of (first column is energy in eV and the second column is 1-sigma fractional energy uncertainty):

```
1.e-5     0.10
100.      0.10
1.e6      0.05
3.e7      0.10
```

then the FUDGE code:

```
>>> XSecMod = XSec * ( 1 + unc )
>>> multiPlot( [ XSec, XSecMod ] )
```

will produce an energy dependent cross section that is 1-sigma above 'XSec' as show in Fig. 2. The plot was generated by the multiPlot function. In this example, the multiplier for sampling of the uncertainty is 1. In general, any sampled value is allowed (e.g., (1 + random * unc)) and, like 'unc', it can be energy dependent.

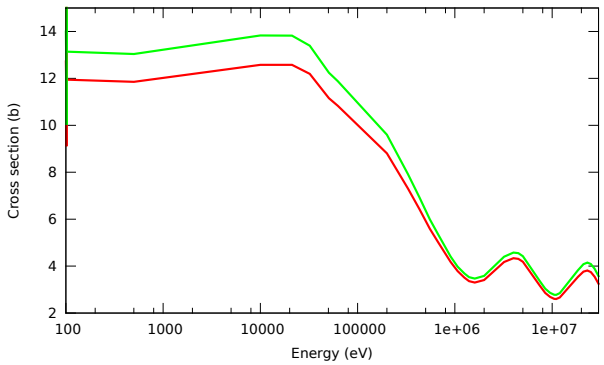

Fig. 2. Plot of the elastic cross section in Fig. 1 (red curve) and the modified cross section (green curve).

Checking Data

Some data issues can only be discovered by comparing transport code results to experiments. However, some issues do not require comparison to experiment to discover. This section discusses the FUDGE checking designed to uncover these latter issues.

For portability and sharing of data as well as other reasons, WPEC sub-group 38 has decided that the default storage for GND will be in XML. For ENDF formatted files, the code 'CHECKR' [10] is used to verify that an ENDF file in properly formatted[c]. For GND/XML files, FUDGE contains an XML schema file [11] that, along with a validation code (e.g., xmllint on a Unix system), checks a GND/XML file for formatting issues.

In addition to checking for format errors, tools to help ensure that a file contains physically realistic data are also needed. For ENDF files, this physics checking has been performed by the combination of two codes, FIZCON and PSYCHE [10]. Together, they check for problems such as unnormalized probability distributions, negative cross sections or multiplicities, non-conservation of energy, incomplete quantities (not covering the full incident energy span required for an evaluation), and so on. Errors uncovered by FIZCON and PSYCHE are reported along with the section (identified by MF and MT numbers) where they occur.

Physics checking is also integrated into the FUDGE implementation of GND. In this case, physics checking is implemented with 'check' methods that are defined in the Python classes. When the 'check' method for a reactionSuite is called, the 'check' method first reconstructs resonances, calculates average outgoing energy for each product, and then proceeds to call the 'check' method of each reaction within the reactionSuite. The 'check' method for a reaction checks its cross section and its list of outgoing products including their distributions and multiplicities. Last, each reaction is checked for energy balance. That is, whether the sum of the average kinetic energies to the products is equal to the available energy (Q-value plus kinetic energy of the projectile) for the reaction.

Using FUDGE to check GND/XML formatted files requires only three lines of Python code, as:

```
>>> from fudge.gnd import reactionSuite
>>> RS = reactionSuite.readXML( "gnd_file.xml" )
>>> print RS.check( )
```

Processing Data

The word "processing" nuclear data has a broad meaning that basically means any transformation of the data. Examples include: reconstructing (see example in **Data Representations and Plotting**) and Doppler broadening cross sections as well as grouping data for deterministic transport. Processing also includes the conversion of GND to other formats (e.g., the ACE format used by the Monte Carlo transport code MCNP - the methods to convert GND to ACE are currently being developed).

FUDGE currently supports most of the processing required to generate nuclear data needed by transport codes, including: reconstructing and Doppler broadening cross sections, calculating average energy for each of the outgoing particle and grouping data for deterministic transport. For deterministic processing, FUDGE is able to

process the entire neutron ENDF/B-VII.1 library with the exception of one reaction. The minimum goal for FUDGE is to support all of the processing currently done by NJOY and AMPX.

Testing Fudge Processing

Rigorous testing of the reconstructed and Doppler broadened cross sections have been performed including comparisons to results from NJOY, RECENT [12] and SIGMA1 [12]. Any issues found (in FUDGE or the other codes) have been fixed and all now agree as expected.

The deterministic data calculated by FUDGE are being compared to those calculated by NJOY and AMPX. Many of the current issues are in the data, not the processing codes. For example, ENDF stores some data redundantly (like masses and spins) and these values often differ. Hence, the result of a calculation depends on which value is used. The testing has uncovers bugs in NJOY and AMPX, and all bugs are reported to the respective code team.

ENDNOTES

[a]GND is not an actual format but an outline (or structure) of how data are to be represented. FUDGE also contains a python script that translates a GND/XML file into a binary GND/HDF5 [13] file.

[b]Python examples are for Python version 2.7.

[c]Some people question whether a new format - or processing toolkit - is needed. However it is worth noting that during the development of the FUDGE translation of ENDF to GND, many bugs were found in the ENDF/B-VII.0 library that were either not uncovered by the ENDF checking codes or were being ignored by the evaluators, but are somehow handled, correctly or not, by the ENDF processing codes. In fact, around 1/4 of the materials in the ENDF/B-VII.0 evaluation cannot be translated by FUDGE since it adheres to the ENDF-6 format documentation [1]. All but about 5 of these issues were reported and fixed before the release of ENDF/B-VII.1. While fewer, other bugs with the data are being uncovered as a result of the testing of FUDGE's processing. Many of these bugs are a result of the ENDF format which stores data redundantly and which the ENDF checker codes are not catching when a discrepancy exists. In part, GND avoids this trap by only storing each datum once.

REFERENCES

1. A. TRKOV, M. HERMAN and D. BROWN (eds.), "ENDF-6 Formats Manual: Data Formats and Procedures for the Evaluated Nuclear Data Files ENDF/B-VI and ENDF/B-VII", BNL-90365-2009 Rev.2, CSEWG Document ENDF-102 (2012). (see http://www.nndc.bnl.gov/csewg/docs/endf-manual.pdf).
2. R.E. MACFARLANEF, A.C. KAHLER, "Methods for Processing ENDF/B-VII with NJOY", *Nuclear Data Sheets*, **111**, *12*, 2739 (2010)
(see http://t2.lanl.gov/codes/njoy99/).
3. M. E. DUNN and N. M. GREENE, "AMPX-2000: A Cross-Section Processing System for Generating Nuclear Data for Criticality Safety Applications", *Trans. Am. Nucl. Soc*. **86**, 118-119 (2002).
4. For information on MCNP see https://mcnp.lanl.gov/.
5. https://www.oecd-nea.org/science/wpec/sg38/
6. B. BECK, et al., "ASCII Format Specifications for the Evaluated Nuclear Data Libraries (ENDL)", UCRL-TM-218475 (2006).
7. D. A, BROWN, et al., "User's Guide to Fete: From ENDF To ENDL", UCRL-SM-218496 (2006).
8. C.M. MATTOON, *et al.*, "Generalized Nuclear Data: A New Structure (with Supporting Infrastructure) for Handling Nuclear Data", *Nuclear Data Sheets*, **113**, 3145-3171 (2012).
9. M.B. CHADWICK, et al., "ENDF/B-VII.1: Nuclear Data for Nuclear Science and Technology: Cross Sections, Covariances, Fission Product Yields and Decay Data", *Nuclear Data Sheets*, **112**, 2887 (2011).
10. C.L. DUNFORD, ENDF Utility Codes Release 7.01/02, Released April 2005.
11. C.E. CAMPBELL, et al., "XML Schema", ACM SIGMOD Record **32** *2* (2003).
12. D. CULLEN, "PREPRO 2010: 2010 ENDF/B Pre-Processing Codes", IAEA Report IAEA-NDS-39, Rev. 14 (2010).
13. The HDF Group. Hierarchical Data Format version 5, 2000-2010, http://www.hdfgroup.org/HDF5.

Acknowledgement: This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.